1. Задача

Інтерполювати поліномами Ньютона, Лагранжа та сплайнами на деякому відрізку числової осі у деякій кількості точок наступну функцію:

$$y=\ln\left(\frac{x^2}{\sqrt{1-8x^4}}\right)$$

Зазначимо, що для отримання коректних результатів обиратимемо відрізок інтерполяції таким чином, щоб $|x|<\sqrt{\sqrt{1/8}}$, x > 0 у всіх його точках. Наведемо вираз для шостої похідної:

$$\frac{d^6}{dx^6}\left(\log\left(\frac{x^2}{\sqrt{1-8x^4}}\right)\right)=$$

$$\frac{240\left(4\,128\,768\,x^{20}+2\,174\,976\,x^{16}+182\,272\,x^{12}+384\,x^8+48\,x^4-1\right)}{x^6\left(1-8\,x^4\right)^6}$$

2. Лістинг програми

```
#include <math.h>
#include <fstream>
#include <iostream>
#include <string>
using namespace std;

long double task (double t) {
	return log(t*t/(sqrt(1-8*t*t*t*t)));
}

double *GausssFunc(int dim, double **A, double *p) { //Ax=p
				int i, j, k, l;
				double koef;
		double max_el;
		double **matr;
		double *q=new double [dim];
		matr=new double *[dim];
				for(i=0; i<dim; i++) {
					matr[i]=new double[dim+1];
				}
				for(i=0; i<dim; i++)
		for(j=0; j<dim+1; j++) {
				if (j!=dim) matr[i][j]=A[i][j];
				else matr[i][j]=p[i];
	}
				for(k=0; k<dim; k++) {
		max_el=matr[k][k];
				for(i=k; i<dim; i++) if (fabs(matr[i][k])>max_el) max_el=fabs(matr[i][k]);
				if (max_el==0) return 0;
					for(l=k+1; l<dim; l++) {
					koef=matr[l][k]/matr[k][k];
				for(j=0; j<dim+1; j++)
					matr[l][j]+=-matr[k][j]*koef;
		}
```

```
		}
		q[dim-1]=matr[dim-1][dim]/matr[dim-1][dim-1];
			for(i=dim-2; i>=0; i--) {
			q[i]=matr[i][dim]/matr[i][i];
			for(j=i+1; j<dim; j++) q[i]+=-matr[i][j]/matr[i][i]*q[j];
			}
			return q;
		}

class Interpolator {
	public:
		double begin;
		double end;
		int n; //n+1 nods
		double *x; //nod's x
		double *y; //nod's y
		double *f; //coefs of Newton's polymon
		double **cub; //for cubic spline
		Interpolator(int N) {
			int i;
			cout<<"Input the beginning of interpolation: ";
			do {
				i=scanf("%lf", &begin);
				fflush(stdin);
				if (i!=1) cout<<"Error, enter value again: ";
			} while(i!=1);
			cout<<"Input the ending if interpolation: ";
			do {
				i=scanf("%lf", &end);
				fflush(stdin);
				if (i!=1) cout<<"Error, enter value again: ";
			} while(i!=1);
			n=N;
			x=new double[n+1];
			y=new double[n+1];
			f=new double[n+1];
			cub=new double*[n];
			for (i=0; i<n; i++) cub[i]=new double[3];
		}
		void Transfer() {
			double a;
			a=begin; begin=end; end=a;
		}
		void GetNet(long double q(double)) {
			int i;
			double step=(end-begin)/n;
			double tx=begin;
			for (i=0; i<n+1; i++) {
				x[i]=tx;
				y[i]=q(tx);
				tx+=step;
			}
			x[n]=end; y[n]=q(end);
		}
		void NewtonInterpolator() {//creates Newt Intpltr ans solves the system
			int i,j=0,k;
			double **fcc;
			fcc=new double *[n+1];
			for(i=0; i<n+1; i++) fcc[i]=new double [n+1];
			for (i=0; i<n+1; i++) {
				fcc[i][0]=1;
				for (j=0; j<n+1; j++) {
					if (i<j) fcc[i][j]=0;
					if (i>0) fcc[i][1]=x[i]-x[0];
					if (i>1) for (k=2; k<i+1; k++) {
```

```
                                        fcc[i][k]=(x[i]-x[k-1])*fcc[i][k-1];
                                }
                        }
                }
                f=GausssFunc(n+1,fcc,y);
}
double Newton(double t) {
        int i, j;
        double S=f[0],P;
        for (i=1; i<n+1; i++) {
                for (j=0, P=f[i]; i>j; j++) P*=(t-x[j]);
                S+=P;
        }
        return S;
}
double Lagrange(double t) {
        int i,j;
        double S, P1, P2;
        for (i=0; i<n+1; i++) {
                for (j=0, P1=y[i], P2=1; j<n+1; j++)
                        if (i!=j) {
                                P1*=(t-x[j]);
                                P2*=(x[i]-x[j]);
                        }
                S+=P1/P2;
        }
        return S;
}
void CubicSplineInterpolator() {//finds coeff of eb cub functions
        int i,j;
        double *h=new double[n];
        double *r=new double[n+1];
        double *p=new double[n+1];
        double **s;
        s=new double *[n+1];
        for(i=0; i<n+1; i++) s[i]=new double[n+1];
        for (i=0; i<n; i++) h[i]=x[i+1]-x[i];
        for (i=0; i<n+1; i++) {//formula(to solve equ)
                if ((i==0) or (i==n)) r[i]=0;
                else r[i]=((y[i+1]-y[i])/h[i]-(y[i]-y[i-1])/h[i-1])*6;
        }
        for (i=0; i<n+1; i++) {//3diagonal matrix
                for (j=0; j<n+1; j++) {
                        if (i==j) {
                                if ((i!=0)&&(i!=n)) s[i][j]=2*(h[i-1]+h[i]);
                                else s[i][j]=1;
                        }
                else if (j==i-1) {
                        if (i!=n) s[i][j]=h[i-1];
                        else s[i][j]=0;
                }
                else if (j==i+1) {
                        if (i!=0) s[i][j]=h[i];
                        else s[i][j]=0;
                }
                else s[i][j]=0;
                }
        }//////////////////////
        r=GausssFunc(n+1,s,r);
        for(i=0; i<n; i++) {
                cub[i][1]=r[i+1]*0.5;
                cub[i][2]=(r[i+1]-r[i])/(h[i]*6);
                cub[i][0]=(h[i]*cub[i][1]-h[i]*h[i]*cub[i][2]+(y[i+1]-y[i])/h[i]);
        }
}
```

```
		long double CubicSpline(double t) {//finds f vls in points t
			int i;
			if (x[0]<x[n]) {
				if (t<x[0]) i=0;
				else if (t>=x[n]) i=n-1;
				else i=(t-x[0])*n/(x[n]-x[0]);
			}
			else {
				if (t>x[0]) i=0;
				else if (t<=x[n]) i=n-1;
				else i=(t-x[0])*n/(x[n]-x[0]);

			}
			double dx=t-x[i+1];
			return cub[i][2]*dx*dx*dx+cub[i][1]*dx*dx+cub[i][0]*dx+y[i+1];
		}
		void Out(long double q(double), string s) {
			double tx=x[0], k;
			double step=(end-begin)/(n);
			ofstream fout(s,ios::out);
			fout << "|  x  |  F(x)   |(Newton-F)(x)|(Lagrange-F)(x)|(CubSpline-F)(x)|" <<
endl;
			do {
				fout <<"|";
				fout.precision(3);
				fout.width(5);
				fout << tx << "|";
				fout.precision(6);
				fout.width(9);
				fout << q(tx) << "|";
				fout.precision(6);
				fout.width(13);
				fout << fixed << fabs(Newton(tx)-q(tx)) << "|";
				fout.precision(6);
				fout.width(15);
				fout << fixed << fabs(Lagrange(tx)-q(tx)) << "|";
				fout.precision(6);
				fout.width(16);
				fout << fixed << fabs(CubicSpline(tx)-q(tx)) << "|";
				fout << endl;
				tx+=step;
			} while (((tx<x[n]) && (step>0))||((step<0)&&(tx>x[n])));
			tx=x[n];
			fout <<"|";
				fout.precision(3);
				fout.width(5);
				fout << tx << "|";
				fout.precision(6);
				fout.width(9);
				fout << q(tx) << "|";
				fout.precision(6);
				fout.width(13);
				fout << fixed << fabs(Newton(tx)-q(tx)) << "|";
				fout.precision(6);
				fout.width(15);
				fout << fixed << fabs(Lagrange(tx)-q(tx)) << "|";
				fout.precision(6);
				fout.width(16);
				fout << fixed << fabs(CubicSpline(tx)-q(tx)) << "|";
				fout << endl;
			fout <<
"#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#
@#@#@#@#@#@" << endl;
			tx = x[0]; step=step/5;  int i=0;
			do {
```

```
                                        if (i==5*n) tx=x[n];
                                        fout <<"|";
                                        fout.precision(3);
                                        fout.width(5);
                                        fout << tx << "|";
                                        fout.precision(6);
                                        fout.width(9);
                                        fout << q(tx) << "|";
                                        fout.precision(6);
                                        fout.width(13);
                                        fout << fixed << fabs(Newton(tx)-q(tx)) << "|";
                                        fout.precision(6);
                                        fout.width(15);
                                        fout << fixed << fabs(Lagrange(tx)-q(tx)) << "|";
                                        fout.precision(6);
                                        fout.width(16);
                                        fout << fixed << fabs(CubicSpline(tx)-q(tx)) << "|";
                                        fout << endl;
                                        tx+=step;
                                        i++;
                                } while (((tx<x[n]) && (step>0))||((step<0) && (tx>x[n])));
                                tx=x[n];
                                fout <<"|";
                                fout.precision(3);
                                fout.width(5);
                                fout << tx << "|";
                                fout.precision(6);
                                fout.width(9);
                                fout << q(tx) << "|";
                                fout.precision(6);
                                fout.width(13);
                                fout << fixed << fabs(Newton(tx)-q(tx)) << "|";
                                fout.precision(6);
                                fout.width(15);
                                fout << fixed << fabs(Lagrange(tx)-q(tx)) << "|";
                                fout.precision(6);
                                fout.width(16);
                                fout << fixed << fabs(CubicSpline(tx)-q(tx)) << "|";
                                fout << endl;
                        }
};

int main() {
        Interpolator B(5);
        B.GetNet(task);
        B.NewtonInterpolator();
        B.CubicSplineInterpolator();
        B.Out(task,"output_up.txt");
        cout<<endl;
        B.Transfer();
        B.GetNet(task);
        B.NewtonInterpolator();
        B.CubicSplineInterpolator();
        B.Out(task,"output_down.txt");
}
```

3. Результати роботи програми

Задану функцію інтерполюємо на відрізку [0.15, 0.3] з кількістю секцій розбиття N=5.

«ВПЕРЕД»:

```
|  x  |  F(x)   |(Newton-F)(x)|(Lagrange-F)(x)|(CubSpline-F)(x)|
| 0.15| -3.79221|     0.000000|       0.000000|        0.000000|
|0.180|-3.425380|     0.000000|       0.000000|        0.000000|
|0.210|-3.113455|     0.000000|       0.000000|        0.000000|
|0.240|-2.840782|     0.000000|       0.000000|        0.000000|
|0.270|-2.596944|     0.000000|       0.000000|        0.000000|
|0.300|-2.374448|     0.000000|       0.000000|        0.000000|
#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#
@#@#@#@#@#@
|0.150|-3.792211|     0.000000|       0.000000|        0.000000|
|0.156|-3.713424|     0.000042|       0.000042|        0.003051|
|0.162|-3.637555|     0.000046|       0.000046|        0.003776|
|0.168|-3.564386|     0.000032|       0.000032|        0.002986|
|0.174|-3.493720|     0.000015|       0.000015|        0.001471|
|0.180|-3.425380|     0.000000|       0.000000|        0.000000|
|0.186|-3.359207|     0.000009|       0.000009|        0.000828|
|0.192|-3.295054|     0.000012|       0.000012|        0.001042|
|0.198|-3.232791|     0.000010|       0.000010|        0.000836|
|0.204|-3.172295|     0.000006|       0.000006|        0.000417|
|0.210|-3.113455|     0.000000|       0.000000|        0.000000|
|0.216|-3.056170|     0.000005|       0.000005|        0.000238|
|0.222|-3.000345|     0.000007|       0.000007|        0.000308|
|0.228|-2.945891|     0.000007|       0.000007|        0.000262|
|0.234|-2.892729|     0.000004|       0.000004|        0.000146|
|0.240|-2.840782|     0.000000|       0.000000|        0.000000|
|0.246|-2.789980|     0.000005|       0.000005|        0.000140|
|0.252|-2.740255|     0.000008|       0.000008|        0.000239|
|0.258|-2.691547|     0.000009|       0.000009|        0.000266|
|0.264|-2.643795|     0.000007|       0.000007|        0.000195|
|0.270|-2.596944|     0.000000|       0.000000|        0.000000|
|0.276|-2.550942|     0.000010|       0.000010|        0.000311|
|0.282|-2.505738|     0.000021|       0.000021|        0.000619|
|0.288|-2.461284|     0.000028|       0.000028|        0.000772|
|0.294|-2.417536|     0.000024|       0.000024|        0.000617|
|0.300|-2.374448|     0.000000|       0.000000|        0.000000|
```

«НАЗАД»:

x	F(x)	(Newton-F)(x)	(Lagrange-F)(x)	(CubSpline-F)(x)
0.3	-2.37445	0.000000	0.000000	0.000000
0.270	-2.596944	0.000000	0.000000	0.000000
0.240	-2.840782	0.000000	0.000000	0.000000
0.210	-3.113455	0.000000	0.000000	0.000000
0.180	-3.425380	0.000000	0.000000	0.000000
0.150	-3.792211	0.000000	0.000000	0.000000
0.150	-3.792211	0.000000	0.000000	0.000000

#@

0.300	-2.374448	0.000000	0.000000	0.000000
0.294	-2.417536	0.000024	0.000024	0.000617
0.288	-2.461284	0.000028	0.000028	0.000772
0.282	-2.505738	0.000021	0.000021	0.000619
0.276	-2.550942	0.000010	0.000010	0.000311
0.270	-2.596944	0.000000	0.000000	0.000000
0.264	-2.643795	0.000007	0.000007	0.000195
0.258	-2.691547	0.000009	0.000009	0.000266
0.252	-2.740255	0.000008	0.000008	0.000239
0.246	-2.789980	0.000005	0.000005	0.000140
0.240	-2.840782	0.000000	0.000000	0.000000
0.234	-2.892729	0.000004	0.000004	0.000146
0.228	-2.945891	0.000007	0.000007	0.000262
0.222	-3.000345	0.000007	0.000007	0.000308
0.216	-3.056170	0.000005	0.000005	0.000238
0.210	-3.113455	0.000000	0.000000	0.000000
0.204	-3.172295	0.000006	0.000006	0.000417
0.198	-3.232791	0.000010	0.000010	0.000836
0.192	-3.295054	0.000012	0.000012	0.001042
0.186	-3.359207	0.000009	0.000009	0.000828
0.180	-3.425380	0.000000	0.000000	0.000000
0.174	-3.493720	0.000015	0.000015	0.001471
0.168	-3.564386	0.000032	0.000032	0.002986
0.162	-3.637555	0.000046	0.000046	0.003776
0.156	-3.713424	0.000042	0.000042	0.003051
0.150	-3.792211	0.000000	0.000000	0.000000

$$\ln\left(\frac{x^2}{\sqrt{1-8x^4}}\right)$$

2 $x = 0.15$

-10 10

3 $x = 0.3$

-10 10

4 $$\frac{240\left(4128768a^{20}+2174976a^{16}+182272a^{12}+384a^{8}+48a^{4}-1\right)}{\left(a-8a^{5}\right)^{6}}$$

= 979474.561424

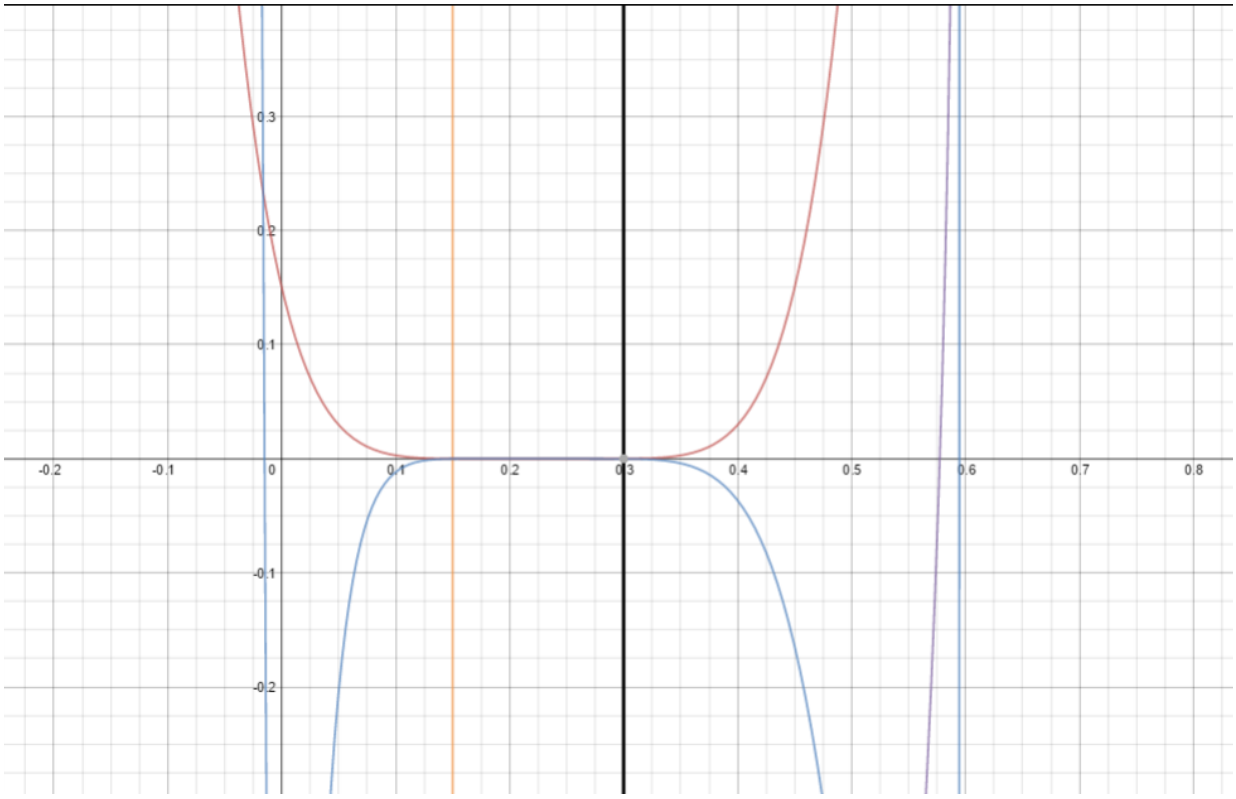
5 $a = 0.398$

0.1 0.5

6 $$\frac{979474}{6!}(x-0.3)(x-0.27)(x-0.24)(x-0.21)(x-0.18)(x-0.15)$$

7 $$\ln\left(\frac{x^2}{\sqrt{1-8x^4}}\right)-\left(-3.79221+12.2277(x-0.15)-30.5032(x-0.15)(x-0.18)+96.6268(x-0.15)(x-0.18)(x-0.21)-269.318(x-0.15)(x-0.18)(x-0.21)(x-0.24)+791.841(x-0.27)(x-0.24)(x-0.21)(x-0.18)(x-0.15)\right)$$

8 $$\ln\left(\frac{x^2}{\sqrt{1-8x^4}}\right)-\left(-2.37445+7.41652(x-0.3)-11.8571(x-0.3)(x-0.27)+46.2436(x-0.3)(x-0.27)(x-0.24)-150.542(x-0.3)(x-0.27)(x-0.24)(x-0.21)+791.841(x-0.3)(x-0.27)(x-0.24)(x-0.21)(x-0.18)\right)$$



purple – our function F(x)
orange, **black** – limits
red – sup function
blue - (Lagrange - F)(x)

4. Висновки:

Ми навчилися інтерполювати функції поліномами Ньютона вперед та назад, Лагранжа (який є іншим записом полінома Ньютона), та кубічними сплайнами. Бачимо, що точність інтерполювання поліномами Ньютона і Лагранжа набагато краща за точністю ніж інтерполювання кубічними сплайнами.